```
@ Filename: agarwal5.s
@ Author:   Vasu Agarwal - vra0004@uah.edu - CS 413-01 spring 2023
@ Purpose:  The objective of this assignment is to simulate the operation of a
vending machine.
@     The machine will dispense, upon reception of the correct amount of money, a
choice of Gum,
@     Peanuts, Cheese Crackers, or M&Ms. Also displaying output using the LED's and
buttons.

@-------------------------------------------------------------------------------------
---
@ Use these commands to assemble, link, run and debug this program:
@   First run the following to get superuser access.
@   "sudo su" is the command to allow running without having to
@    use sudo.
@    as -o agarwal5.o agarwal5.s
@    gcc -o agarwal5 agarwal5.o -lwiringPi
@    ./agarwal5 ;echo $?
@    gdb --args ./agarwal5
@-------------------------------------------------------------------------------------
----

@ 's' is secret inventory code




OUTPUT = 1 @ Used to set the selected GPIO pins to output only.

ON = 1 @ Turn the LED on.

OFF = 0 @ Turn the LED off.

RED = 5 @ Pin number from wiringPi for red led

YELLOW = 4 @ Pin number from wiringPi for yellow led

GREEN = 3 @ Pin number from wiringPi for green led

BLUE = 2 @ Pin number from wiringPi for blue led


@ Define the following from wiringPi.h header

INPUT  = 0

PUD_UP   = 2

PUD_DOWN = 1

LOW  = 0

HIGH = 1


.equ READERROR, 0
```

```
.global main
main:
      @ check the setup of the GPIO to make sure it is working right.
      @ To use the wiringPiSetup function just call it on return:
      @ r0 - contains the pass/fail code
      bl wiringPiSetup
      mov r1,#-1
      cmp r0, r1
      bne init @ Everything is OK so continue with code.
      ldr r0, =ErrMsg
      bl printf
      b errorout @ There is a problem with the GPIO exit code.
init:
@ set the mode to input - BLUE
    ldr     r0, =buttonBlue
    ldr     r0, [r0]
    mov     r1, #INPUT
    bl      pinMode
@ set the mode to input - GREEN
    ldr     r0, =buttonGreen
    ldr     r0, [r0]
    mov     r1, #INPUT
    bl      pinMode
@ set the mode to input- YELLOW
    ldr     r0, =buttonYellow
    ldr     r0, [r0]
    mov     r1, #INPUT
    bl      pinMode
@ set the mode to input - RED
```

```
ldr     r0, =buttonRed
ldr     r0, [r0]
mov     r1, #INPUT
bl      pinMode

  @ set the blue LED mode to output
  ldr r0, =blue_LED
  ldr r0, [r0]
  mov r1, #OUTPUT
  bl pinMode
  @ set the green LED mode to output
  ldr r0, =green_LED
  ldr r0, [r0]
  mov r1, #OUTPUT
  bl pinMode
  @ set the yellow LED mode to output
  ldr r0, =yellow_LED
  ldr r0, [r0]
  mov r1, #OUTPUT
  bl pinMode
  @ set the red LED mode to output
  ldr r0, =red_LED
  ldr r0, [r0]
  mov r1, #OUTPUT
  bl pinMode
  @ Write a logic one to turn pin to on.
  ldr r0, =red_LED
  ldr r0, [r0]
  mov r1, #ON
  bl digitalWrite
```

```
      ldr r0, =delayMs
      ldr r0, [r0]
      bl delay
      @ Write a logic 0 to turn pin5 off.
      ldr r0, =red_LED
      ldr r0, [r0]
      mov r1, #OFF
      bl digitalWrite



@ Setup and read all the buttons.
@ Set the buttons for pull-up and it is 0 when pressed.
@    pullUpDnControl(buttonPin, PUD_UP)
@    digitalRead(buttonPin) == LOW button pressed
    ldr  r0, =buttonBlue
    ldr  r0, [r0]
    mov  r1, #PUD_UP
    BL   pullUpDnControl

    ldr  r0, =buttonGreen
    ldr  r0, [r0]
    mov  r1, #PUD_UP
    BL   pullUpDnControl

    ldr  r0, =buttonYellow
    ldr  r0, [r0]
    mov  r1, #PUD_UP
    BL   pullUpDnControl
    ldr  r0, =buttonRed
    ldr  r0, [r0]
    mov  r1, #PUD_UP
    BL   pullUpDnControl

setCounts:
      mov r4, #2 @ gum count
      mov r5, #2 @ peanuts count
      mov r6, #2 @ crackers count
      mov r7, #2 @ m&ms count


    mov r9,  #0xff
    mov r10,  #0xff
    mov r11, #0xff
    mov r12, #0xff
prompt:
      ldr r0, =strInputPrompt @ welcomes user and provides instructions
      bl printf

ButtonLoop:


@ Delay a few miliseconds to help debounce the switches.
@
    ldr  r0, =delay25Ms
    ldr  r0, [r0]
    BL   delay

ReadBLUE:
@ Read the value of the blue button. If it is HIGH (i.e., not
@ pressed) read the next button and set the previous reading
@ value to HIGH.
@ Otherwise the current value is LOW (pressed). If it was LOW
@ that last time the button is still pressed down. Do not record
@ this as a new pressing.
@ If it was HIGH the last time and LOW now then record the
@ button has been pressed.
@
    ldr    r0,  =buttonBlue
    ldr    r0,  [r0]
    BL     digitalRead
    cmp    r0, #HIGH
    moveq  r9, r0
    beq    ReadGREEN

    cmp    r9, #LOW
    beq    ReadGREEN

    mov    r9, r0
    b      PedBLUE

ReadGREEN:
@ See comments on BLUE button on how this code works.
@
    ldr    r0,  =buttonGreen
    ldr    r0,  [r0]
    BL     digitalRead
    cmp    r0, #HIGH
    moveq  r10, r0
    beq    ReadYELLOW
    cmp    r10, #LOW
    beq    ReadYELLOW

    mov    r10, r0
    b      PedGREEN

ReadYELLOW:
@ See comments on BLUE button on how this code works.
@
    ldr    r0,  =buttonYellow
    ldr    r0,  [r0]
    BL     digitalRead
    cmp    r0, #HIGH
    moveq  r11, r0
    beq    ReadRED

    cmp    r11, #LOW
    beq    ReadRED

    mov    r11, r0
    b      PedYELLOW

ReadRED:
@ See comments on BLUE button on how this code works.
@
    ldr    r0,  =buttonRed
    ldr    r0,  [r0]
    BL     digitalRead
    cmp    r0, #HIGH
    moveq  r12, r0
    beq    ButtonLoop

    cmp    r12, #LOW
    beq    ButtonLoop

    mov    r12, r0
    b      PedRED

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ Printing out which button was pressed.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

PedBLUE:
    LDR  r0, =PressedBLUE @ Put address of string in r0
    BL   printf           @ Make the call to printf
    B    mnms       @ Go read more buttons



PedGREEN:
    LDR  r0, =PressedGREEN @ Put address of string in r0
    BL   printf            @ Make the call to printf
    B    crackers        @ Go read more buttons

PedYELLOW:
    LDR  r0, =PressedYELLOW @ Put address of string in r0
    BL   printf             @ Make the call to printf
    B    peanuts           @ Go read more buttons
```

```
PedRED:
    LDR  r0, =PressedRED  @ Put address of string in r0
    BL   printf           @ Make the call to printf
    B    gum                @ Go read more buttons



gum:
      ldr r0, =userSelection
      ldr r1, =candyG
      bl printf @ prints the confirmation question (y/n)
      ldr r0, =userInput
      ldr r1, =numInput
      bl scanf @ reads in the y/n from the user
      cmp r0, #READERROR

      beq readerror
      ldr r1, =numInput
      ldr r1, [r1]
      cmp r1, #'n'
      beq prompt @ if n or anything else, go back to prompt
      cmp r1, #'s'
      beq printinventory
      cmp r4, #0
      beq emptyinv @ branches to print that there are no more ininventory
      sub r4, r4, #1 @ otherwise, subtracts from inventory
inventoryG:
      mov r1, #50
      push {r1} @ pushes the cost of gum (50 cents)
      b popper
```

```
peanuts:
      ldr r0, =userSelection
      ldr r1, =candyP
      bl printf
      ldr r0, =userInput
      ldr r1, =numInput
      bl scanf
      cmp r0, #READERROR
      beq readerror
      ldr r1, =numInput
      ldr r1, [r1]
      cmp r1, #'n'
      beq prompt @ if n or anything else, go back to prompt
      cmp r1, #'s'
      beq printinventory
      cmp r5, #0
      beq emptyinv
      sub r5, r5, #1
inventoryP:
      mov r1, #55
      push {r1}
      b popper
crackers:
      ldr r0, =userSelection
      ldr r1, =candyC
      bl printf
      ldr r0, =userInput
      ldr r1, =numInput
      bl scanf
      cmp r0, #READERROR
      beq readerror
      ldr r1, =numInput
      ldr r1, [r1]
      cmp r1, #'n'
      beq prompt @ if n or anything else, go back to prompt
      cmp r1, #'s'
      beq printinventory
      cmp r6, #0
      beq emptyinv
      sub r6, r6, #1
inventoryC:
      mov r1, #65
      push {r1}
      b popper
mnms:
      ldr r0, =userSelection
      ldr r1, =candyM
      bl printf
      ldr r0, =userInput
      ldr r1, =numInput
      bl scanf
      cmp r0, #READERROR
      beq readerror
      ldr r1, =numInput
      ldr r1, [r1]
      cmp r1, #'n'
      beq prompt @ if n or anything else, go back to prompt
      cmp r1, #'s'
      beq printinventory
```

```
    cmp r7, #0
    beq emptyinv
    sub r7, r7, #1
    inventoryM:
    mov r1, #100
    push {r1}
    b popper
emptyinv:
    ldr r0, =noInventory
    bl printf @ prints that there is no inventory left for an item
    b prompt
popper:
    pop {r8} @ pops the original price of an item into r8
    push {r11}
    mov r11, r8 @ puts a copy in r11
inventory:
    ldr r0, =userPayment
    mov r1, r8
    bl printf @ prompts the user to enter x cents
    ldr r0, =userInput
    ldr r1, =numInput
    bl scanf @ reads in (D, Q, B) as the change entered
    cmp r0, #READERROR
    beq readerror
    ldr r1, =numInput
    ldr r1, [r1]
    cmp r1, #'D'
    beq dime
    cmp r1, #'Q'
    beq quarter
```

```
      cmp r1, #'B'

      beq dollarbill

dime:

      sub r8, r8, #10 @ subtracts 10 cents if a dime is entered

      cmp r8, #0 @ if the total cost remaining has reached zero, end loop

      ble change

      b inventory @ else, continue loop

      quarter:

      sub r8, r8, #25

      cmp r8, #0

      ble change

      b inventory

dollarbill:

      sub r8, r8, #100

      cmp r8, #0

      ble change

      b inventory

change:

      ldr r0, =enoughPayment

      bl printf @ informs user that enough payment has been provided

      cmp r11, #50

      beq printgum

      cmp r11, #55

      beq printpeanuts

      cmp r11, #65

      beq printcrackers

      cmp r11, #100

      beq printmnms

printgum:
      pop {r11}
      push {r9}
      mov r9, #3
forLoop:
      ldr r0, =red_LED
      ldr r0, [r0]
      mov r1, #ON
      bl digitalWrite
      ldr r0, =delay1Ms
      ldr r0, [r0]
      bl delay
      ldr r0, =red_LED
      ldr r0, [r0]
      mov r1, #OFF
      bl digitalWrite
      ldr r0, =delay1Ms
      ldr r0, [r0]
      bl delay

      sub r9, #1
      cmp r9, #0
      bne forLoop
      ldr r0, =red_LED
      ldr r0, [r0]
      mov r1, #ON
      bl digitalWrite
      ldr r0, =delayMs
      ldr r0, [r0]
      bl delay
      ldr r0, =red_LED
      ldr r0, [r0]
      mov r1, #OFF
      bl digitalWrite
      ldr r0, =dispensed
      ldr r1, =candyG
      bl printf @ prints that item has been successfully dispensed
      pop {r9}
      b changeoutput
printpeanuts:
      pop {r11}
      push {r9}
      mov r9, #3
forLoop1:
      ldr r0, =yellow_LED
      ldr r0, [r0]
      mov r1, #ON
      bl digitalWrite
      ldr r0, =delay1Ms
      ldr r0, [r0]
      bl delay
      ldr r0, =yellow_LED
      ldr r0, [r0]
      mov r1, #OFF
      bl digitalWrite
      ldr r0, =delay1Ms
      ldr r0, [r0]
      bl delay
      sub r9, #1
      cmp r9, #0
      bne forLoop1

      ldr r0, =yellow_LED

      ldr r0, [r0]

      mov r1, #ON

      bl digitalWrite

      ldr r0, =delayMs

      ldr r0, [r0]

      bl delay

      ldr r0, =yellow_LED

      ldr r0, [r0]

      mov r1, #OFF

      bl digitalWrite

      ldr r0, =dispensed

      ldr r1, =candyP

      bl printf

      pop {r9}

      b changeoutput

printcrackers:

      pop {r11}

      push {r9}

      mov r9, #3

forLoop2:

      ldr r0, =green_LED

      ldr r0, [r0]

      mov r1, #ON

      bl digitalWrite

      ldr r0, =delay1Ms

      ldr r0, [r0]

      bl delay

      ldr r0, =green_LED
      ldr r0, [r0]

      mov r1, #OFF

      bl digitalWrite

      ldr r0, =delay1Ms

      ldr r0, [r0]

      bl delay

      sub r9, #1

      cmp r9, #0

      bne forLoop2

      ldr r0, =green_LED

      ldr r0, [r0]

      mov r1, #ON

      bl digitalWrite

      ldr r0, =delayMs

      ldr r0, [r0]

      bl delay

      ldr r0, =green_LED

      ldr r0, [r0]

      mov r1, #OFF

      bl digitalWrite

      ldr r0, =dispensed

      ldr r1, =candyC

      bl printf

      pop {r9}

      b changeoutput

printmnms:

      pop {r11}

      push {r9}

      mov r9, #3
```

```
forLoop3:
      ldr r0, =blue_LED
      ldr r0, [r0]
      mov r1, #ON
      bl digitalWrite
      ldr r0, =delay1Ms
      ldr r0, [r0]
      bl delay
      ldr r0, =blue_LED
      ldr r0, [r0]
      mov r1, #OFF
      bl digitalWrite
      ldr r0, =delay1Ms
      ldr r0, [r0]
      bl delay
      sub r9, #1
      cmp r9, #0
      bne forLoop3
      ldr r0, =blue_LED
      ldr r0, [r0]
      mov r1, #ON
      bl digitalWrite
      ldr r0, =delayMs
      ldr r0, [r0]
      bl delay
      ldr r0, =blue_LED
      ldr r0, [r0]
      mov r1, #OFF
      bl digitalWrite
      ldr r0, =dispensed
      ldr r1, =candyM

      bl printf

      pop {r9}

      b changeoutput

changeoutput:

      push {r9}

      ldr r0, =changeOutput

      mov r1, r8

      mov r9, #-1 @ makes negaive number positive to represent change

      mul r1, r8, r9

      bl printf @ prints the amount of change returned

      pop {r9}

checkinventory: @ checks the inventory of each item to check if program should
continue

      cmp r4, #0

      bne prompt

      cmp r5, #0

      bne prompt

      cmp r6, #0

      bne prompt

      cmp r7, #0

      bne prompt

      ldr r0, =noInventory

      bl printf

b myexit

printinventory: @ section for secret input

      ldr r0, =secretInventory

      mov r1, r4

      mov r2, r5

      mov r3, r6
      bl printf
      ldr r0, =mnmInventory
      mov r1, r7
      bl printf
      b prompt

readerror:
      ldr r0, =strInputPattern
      ldr r1, =strInputError
      bl scanf
      b prompt
myexit:
      ldr r0, =red_LED
      ldr r0, [r0]
      mov r1, #ON
      bl digitalWrite
      ldr r0, =delayMs
      ldr r0, [r0]
      bl delay
      ldr r0, =red_LED
      ldr r0, [r0]
      mov r1, #OFF
      bl digitalWrite
      mov r7, #0x01
      svc 0
done:
      b myexit
      errorout: @ Label only need if there is an error on board init.
      mov r0, r8
      MOV r7, #0X01
      SVC 0



.data
.balign 4

buttonBlue:   .word 7 @Blue button
buttonGreen:  .word 0 @Green button
buttonYellow: .word 6 @Yellow button
buttonRed:    .word 1 @Red button

delay25Ms: .word 250  @ Delay time in Miliseconds.



.balign 4
PressedBLUE: .asciz "The BLUE button was pressed. \n"
.balign 4
PressedYELLOW: .asciz "The YELLOW button was pressed.\n"
.balign 4
PressedGREEN: .asciz "The GREEN button was pressed. \n"
.balign 4
PressedRED:  .asciz "The RED button was pressed. \n"
.balign 4
ErrMsg: .asciz "Setup didn't work... Aborting...\n"

.balign 4
strInputPrompt: .asciz "\nWelcome to the vending machine.\nGum: $.50, Peanuts: $.55, Cheese Crackers: $.65, M&Ms: $1.00 \nPress a button to select an item (Red, Yellow, Green, Blue)\n"
.balign 4
```

```
userSelection: .asciz "\nYou selected %s. Is this correct (y/n)? \n"

.balign 4

userPayment: .asciz "\nEnter at least %d cents for selection.\
nDimes(D),Quarters(Q), and Dollar Bills(B): \n"

.balign 4

dispensed: .asciz "\n%s has been dispensed.\n"

.balign 4

enoughPayment: .asciz "\nEnough money entered. \n"

.balign 4

noInventory: .asciz "\nOut of Inventory!\n"

.balign 4

secretInventory: .asciz "\nGum - %d \nPeanuts - %d \nCheese Crackers - %d\n"

.balign 4

mnmInventory: .asciz "M&Ms - %d\n"

.balign 4

changeOutput: .asciz "\nChange of %d cents has been returned. \n"

.balign 4

candyG: .asciz "gum"

.balign 4

candyP: .asciz "peanuts"

.balign 4

candyC: .asciz "cheese crackers"

.balign 4

candyM: .asciz "M&Ms"

.balign 4

userInput: .asciz "%s"

.balign 4

strOutputNum: .asciz "%d \n"

.balign 4

strOutputArea: .asciz "\nArea: %d \n"
```

```
.balign 4
numInputPattern: .asciz "%d"
.balign 4
strInputPattern: .asciz "%[^\n]"
.balign 4
strInputError: .skip 100*4
.balign 4
numInput: .word 0

@ Define the values for the pins
blue_LED : .word BLUE
green_LED : .word GREEN
yellow_LED : .word YELLOW
red_LED : .word RED
delayMs: .word 500 @ Set delay for five seconds.
delay1Ms: .word 100
.balign 4
string1: .asciz "Raspberry Pi Blinking Light with Assembly. \n"
.balign 4
string1a: .asciz "This blinks the LEDs on the Board. \n"
.balign 4
string2: .asciz "The four LEDs should have blinked. \n"
.global printf
.global scanf

@ The following are defined in wiringPi.h

.extern wiringPiSetup
.extern delay
.extern digitalWrite
```

```
.extern pinMode
```